IMAGE PROCESSING WORKBENCH : FILTERS AND MORPHOLOGICAL TOOLS

A thesis submitted

in partial fulfilment of the requirements

for the degree of

Master Of Technology

by

Maj AK Nath

to the

Department of Computer Science and Engineering

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

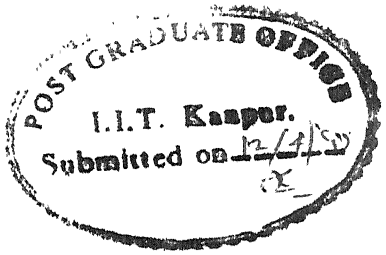
April, 1995

CERTIFICATE

It is certified that the work contained in the thesis entitled *Image Processing Workbench : Filters And Morphological Tools*, by Maj AK Nath, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Harish Karnick
Associate Professor,
Department Of Computer Science
and Engineering,
IIT , Kanpur

April 1995



ABSTRACT

This thesis seeks to provide a workbench for image processing filter functions. Stress has been laid to make the workbench user friendly and easy to operate for users without an intricate knowledge of the algorithms used in the workbench.Various user tools have been provided to manipulate and display the desired images. A lot of flexibility has been provided for backward and forward integration of this workbench to make it a complete image processing system . Frequently used processing and file handling routines are available for later use and modification in the form of dynamic link libraries. This workbench is portable on any machine running MS-Windows 3.1.

ACKNOWLEDGEMENTS

I wish to thank Dr H Karnick for his able guidance during the course of this thesis. His foresight, imagination and patient manner made the work very interesting and stimulating.

I am thankful to all the faculty members of the CSE department who have always taken interest in solving any problems which came up. I am also thankful to the staff of the department for their co-operation .

Last but not the least I value the invisible but important backing of my family, which made my working environment a conducive and enjoyable one.

TABLE OF CONTENTS

List Of Figures

Page

CHAPTER 1

INTRODUCTION

1.1 Background

Interest in digital image processing methods stems from two principal application areas : improvement of pictorial information for human interpretation, and processing of image data for machine perception. Processing of digital images involves procedures that are usually expressed in algorithmic form. Thus most image processing functions can be implemented in software. The need for specialized hardware for processing or storage can be done away with, if the application can do without the inherent speed, higher storage capacity and high resolution display of such hardware. Image processing requirements exist in many areas of science and technology such as defence(satellite imagery), medical imaging, remote sensing, astronomy, metallurgy, civil engineering, chemical engineering etc. Dedicated systems are needed to address the problems typical to these fields. However it is found that till a particular stage, all these systems require processed information of a common type which involves techniques for increasing contrast, noise removal and edge detection etc.

1.2 Aim And Scope

It is with the above in mind, that an attempt has been made in this thesis to develop a workbench for image processing filter functions.

The scope of this thesis encompasses the following :

- Use of a 80386 PC/AT or above with a VGA / SVGA display card and monitor for application development and porting.

- To make use of the limited capabilities of the available PC hardware to produce as faithful an image display as possible .
- To build a robust, flexible and user friendly interface for users from diverse fields, without much formal knowledge of image processing, to be able to prototype an application.
- To make available, the processing and other routines in the form of libraries for incorporation into dedicated applications.

1.3 Organization Of The Thesis

This thesis is organized into four other chapters besides this and three appendices . Chapter 2 deals with the general design and implementation philosophy and choice of the platform for program development. Chapter 3 provides a step by step guide to the operation of the package and an illustration of the various user tools and controls enumerated in chapter 2. Chapter 4 contains the details of the various filter functions implemented. Chapter 5 has a summary, conclusions and pointers to further work. Description of terms used is given in the glossary. A description of the file format used is given in appx 'A'. Appx 'B' contains the standard masks used in the filter functions. Appendix 'C' contains the results obtained with the help of this workbench on a few example images.

CHAPTER 2

DESIGN AND IMPLEMENTATION ASPECTS

The main design issues to be addressed in the thesis were :

- a robust user friendly interface
- the choice of operations to be implemented
- a facility to build prototypes of applications rapidly by composing image processing operations in a flexible manner.

2.1 User Interface

The user interface was further broken down into :

- provision for a quick and versatile display of the selected image on the screen
- design of a menu driven system from where a user selects various operations and interacts with the system for giving in the required inputs.

This led to MS-Windows being chosen as the software platform for the implementation.

2.1.1 Some Unsuccessful Display Approaches

Having gone in for a 386 PC/AT and above, the display hardware posed a serious limitation for a faithful display of the image. Since a plethora of monitors and display adapters are available for the IBM PC and compatibles, a display mechanism which could be portable on most of them chosen for the scope of this thesis was required. For this their resolutions , size of video RAM and palettes available had to be considered. CGA and EGA cards were ruled out because of their low resolution and limited colors/gray levels.

Having selected the VGA and above series of display adapters an attempt was made to directly program the video RAM . The image was acquired from the CCD camera in a flat format i.e. the pixel values represented the intensity in a range of 0 to 255 graylevels.

The low level ROM BIOS video services were used for this purpose DOS Interrupt 10H was used to directly program the video RAM. The various registers on the 80x86 chip were given the initialization data required by the various services provided with interrupt 10H. Ref [EVGA] . Video mode 12H was selected for this purpose but the results were far from satisfactory. This approach was shelved due to the following shortcomings :

- The display came up line by line and the time taken to display the complete image was unacceptably high.
- The aspect ratio of the monitor attached to the PC being different from that of the CCD camera caused distortions in the image.
- The object boundaries appeared jagged, especially if they lay along an inclined plane.
- For the video mode selected (resolution 640X480 for VGA) only 16 gray levels could be displayed This required an algorithm for mapping the 256 gray levels given as output by the CCD camera to the nearest 16 gray levels statistically. The lower resolution mode 13H (320X200 for VGA) was also tried out as it enabled display of 256 gray levels but was not found suitable because of the coarse resolution.
- The image size was limited by the size of the contiguous memory block available as buffer to store the image.(This happens to be the size of the available RAM in case virtual memory is not being implemented by the platform being used)

Another approach using the C graphics library functions for the display was tried out. This time the palette initialization and graylevel mapping was taken care of by the available library functions but a few problems remained and a few more emerged. They are summarized below:

- The display took longer to come up than before.
- Limitation on the image size still remained.

A problem common to both the approaches was that though the screen size did not pose a serious limitation to the maximum permissible size of the image, an efficient scrolling mechanism was required to enable the viewing of the entire image. This would require the display to be refreshed again and again. Considering the time taken for one display scrolling the image would turn out to be a very time consuming job. The above listed disadvantages were enough to discard these two approaches.

Since the above approaches were not found suitable for the display part, their use for designing the user interface was never considered. All these attempts reinforced the choice of the MS Windows environment since it could be used for the display part.

2.1.2 Reasons For Choosing The MS-Windows Environment

For the program developer, Windows provides a wealth of built in routines that allow use of Menus, Dialog boxes, Scroll bars and other components of a graphical point and click user interface. Windows can also run programs written for MS-DOS. These programs can of course not take advantage of Windows features but some of them can be windowed and run with other windows applications.

The user directly interacts with the objects on display with the help of a mouse or the keyboard .Graphical objects can be dragged, buttons can be pushed and scroll bars scrolled to denote various actions inside programs. Windows programs can be run with any video board for which a Windows device driver is available without the need to know the type of device attached.

Windows incorporates a "Message Driven" architecture. Any user action with any of the objects on screen like a menu, window, child window dialog box or a scroll bar generates a "message" which is placed in the application's "message queue". The messages are processed by the application's window procedure or the "window processing function". This function is organized in the form of switch-case statements. Windows has inherent object orientation built in. Every window is an object based on a certain

"windows class". The code it contains is the window processing function and it's data is the information retained inside the function.

Virtual memory implementation and the capability to move and swap memory blocks to and from disk is a Windows feature (Windows 3.x running in 386 enhanced mode) which can be used in buffer allocation for processing large amounts of data.

2.2 Display Design

The Windows features enumerated in the previous section made it the automatic choice for both the display as well as the user interface implementation. Advantage has been taken of the extensive library of Windows "Graphics Device Interface" (GDI) functions available in Windows. The input to the program is in the form of a Windows .BMP file. This is a bitmap file in which pixel values represent indices to the color data given in the header. The entire .BMP format is explained in appx 'A'. Due to a lot of variation in the kinds of .BMP files, a standard format has been chosen for the input. Any file selected to be processed by the program is first read into a buffer and converted to this standard .BMP format given at appendix 'A'. The advantage of choosing this particular type of format lies in the fact that the pixel values, in addition to representing indices into the color table also represent the true graylevels and hence the pixel data can be processed directly without repeated table lookups. This is possible because the values in the color table vary from 0 to 255 in that order for Red, Blue And Green colors together. Table lookup is performed only once during read in time after which, the standard format mentioned above is used for subsequent processing. Pixel values are scaled to a range of 0 to 255 after every operation.

A buffer is always maintained for the original bitmap as well as the processed bitmap. The processed bitmap buffer contains the bitmap obtained after the most recent processing operation. The bitmap is stored with all the header information. It can therefore be saved to a .BMP file at any time. For display the bitmap in the buffer is

selected into a memory device context. The size of the display area or more appropriately the "client area " is then determined. Only the portion of the bitmap fitting into the client area is put on the display device context. While determining the portion of the bitmap to be displayed, the vertical and horizontal scroll bar positions are also taken into account. The image can thus be scrolled with the help of the scroll bars in both directions. This feature is useful in viewing larger than screen images.

The image display on screen is instant and not line by line as experienced in the earlier approaches. This has been made possible by the use of the "memory device context". A block transfer of the image takes place from the memory device context on to the display device. Due to virtual memory implementation by Windows, the image size is not limited by the amount of free memory available but by the free disk space. The upper limit of a contiguous memory block which is allowed by Windows is 16 MB. This is allocated as a "huge" pointer from the global memory heap.

2.3 User Interface Design

The user interface for this workbench has been designed by taking advantage of the message driven architecture of windows and other features mentioned in sec 2.1.2. The interface is menu driven. Menu items invoke the operations required to be performed on the image. There are menus for display manipulation as well as opening and saving the image files. The input for various operations is interactively taken from the user by using child window controls and dialog boxes. The use of the keyboard has been sought to be minimized. Most inputs can be given with the help of the mouse itself.

The selection of a menu item generates a message in the application's "message queue" under an id no. The operation matching the id no. is invoked if the message is processed by the current window's processing function, else it is passed to the default window procedure.

2.3.1 File Opening And Saving

The first action expected from the user is to select and open the .BMP file he wants to view and process. This is done from the menu bar. The standard FILEOPEN data structure (appx 'A') is first initialized. A file open dialog box comes up on screen which prompts the user to either type or select the relevant .BMP filename required by him. It is possible to change drives and directories and select any filename desired. The selected filename is filled into the data structure and also passed to a function which opens the desired file, reads it and returns a bitmap in standard format (appx 'A') in the form of a huge pointer to BYTE. During this process, the following actions take place :

- The selected file is validated for being a .BMP file in uncompressed form.
- In case found to be valid, it's header information is read and interpreted.
- If the file is found not conforming to the required format, an error message is given and the function returns NULL.
- The pixel data of the file is now read and converted into actual intensity values by performing table lookup. Files containing pixel data in 1 bit per pixel or 4 bits per pixel are all converted into a standard 8 bit per pixel, uncompressed format (appx 'A').

Control then passes on to the display function which displays the bitmap in a separate display window.

For saving the processed results a file save dialog box comes up which uses the same data structure as the file open dialog box. The bitmap is saved on disk in the same standard format. The processed bitmap is also available as a huge pointer to BYTE, complete with all the header information. The size of the whole bitmap is available in the header itself and is written on to disk in chunks of 32767 bytes to suit the operating system requirements. Windows takes the help of the underlying DOS for all its file handling and disk input/output.

2.3.2 User Input Interface

Once the bitmap is displayed and selected for processing a filter menu item is selected, generating a message in the application's message queue. The relevant case statement in the window processing function now takes over. The inputs required are taken from the user in dialog boxes and child window controls for carrying out initialization. Each dialog box has it's own "processing function". This can be invoked from the main window processing function by obtaining a handle for it. When the dialog box has the "input focus" all the messages generated are now passed to and processed by the dialog box message processing function. The dialog boxes contain a number of child window controls which are used for taking user inputs. User interaction with these child windows causes generation of messages. The inputs are passed to variables when these messages are processed by the respective processing function of these dialog boxes.

2.3.2 Filter Functions

The filter functions along with the file handling functions are available in the form of dynamic link libraries which are linked with the program during run time. Details of filter implementations are given in chapter 4.

2.4 User Tools

Besides the basic functionalities discussed above, some more features are available to the user, thus adding teeth to the workbench. A step by step pictographic explanation of the same is given in chapter 3. They are enumerated below :

- An option to display the processed image in the processing window in it's true size along with scrolling facility.
- An additional facility to display the processed image at any place on the screen defined by the user in the form of a rubber banded rectangle drawn with the mouse. The whole image fits into this rectangle. This is useful for comparing results.
- Options to clear the whole screen or only selected portions of it.

- Facility to view the original image at any time for comparison by maximizing and re-sizing the display window which is present in minimized form once the processing begins.
- An additional option to select a defined part of the original image for processing. The part to be processed is defined by the user in the form of a rubber banded rectangle.
- A batch programming facility for users familiar with the system. A pre-defined set of masks is used for this to eliminate the time taken for inputs Processing is always carried out on the previously processed bitmap.
- Context sensitive help is available for all windows and input dialog boxes. It can be invoked by clicking the help menu item (for the window menu), HELP button (for dialog boxes) or the F1 and F2 keys. F1 key takes the user to the main index, F2 is the context sensitive key.

CHAPTER 3

A GUIDED TOUR OF THE PACKAGE

In this chapter the operation of the entire workbench has been explained with the help of an example. Actions a user has to take at each step have been described. The various user tools, dialog boxes and child window controls have all been shown pictographically along with the way to give the expected inputs. The filename chosen for the example is chosen as example.bmp

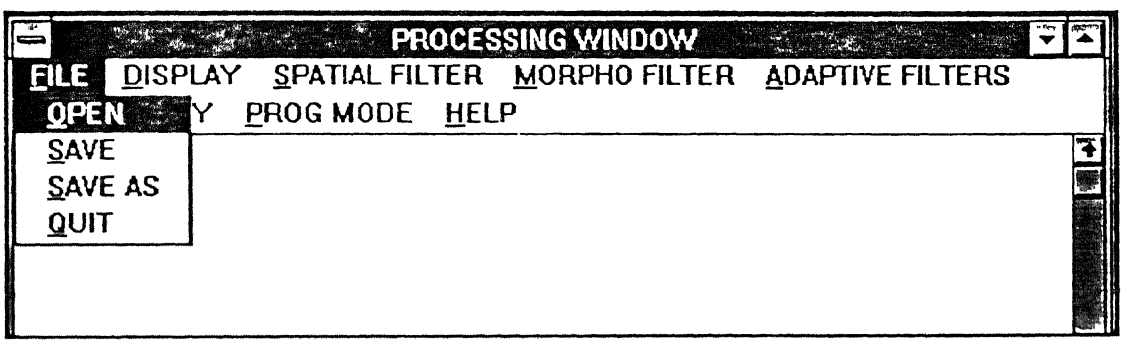


Figure 3.1. Startup

On startup, the Processing window shows up as above from where the user is expected to select the file to open using the FILE | OPEN menu item. The layout of the window is as seen above. The control menu bar is onto the left top from where one can close, maximize, minimize or restore the window or even switch to another application. The caption bar is on the top with the window name. The minimize and maximize / restore buttons are on the right of the caption bar. The portion below the caption bar is the menu bar containing the menu items. The rest of the area is the client area where the display takes place.

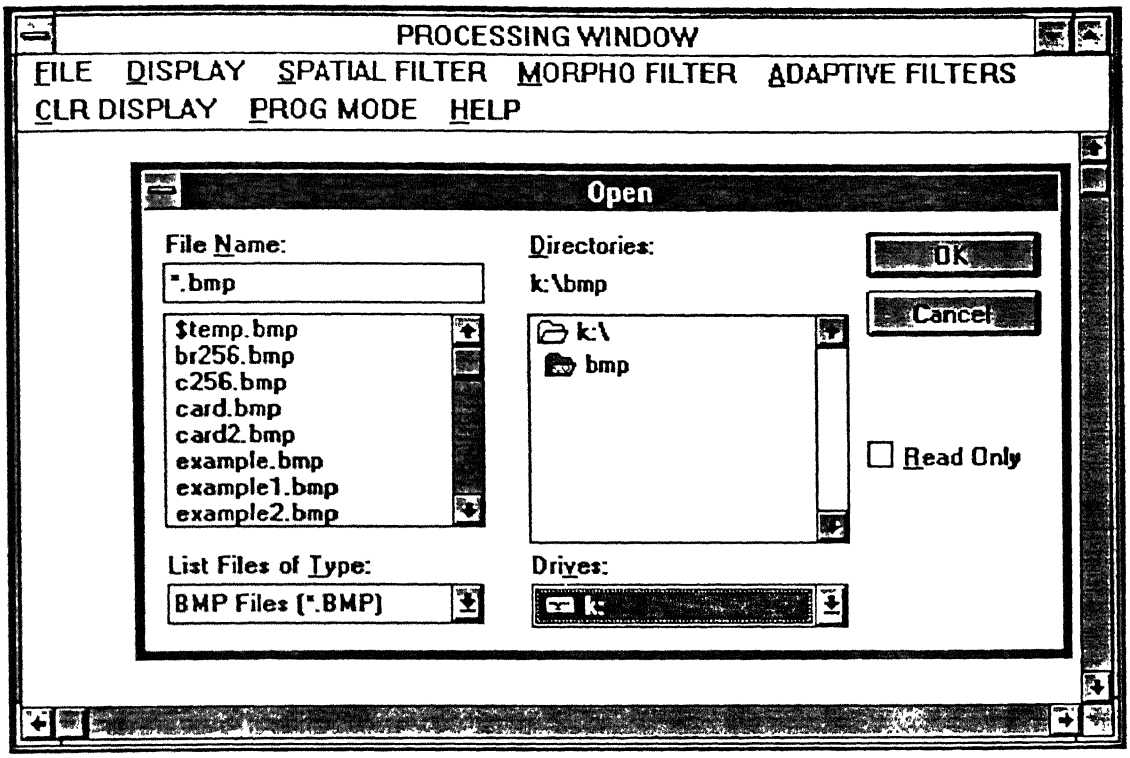


Figure 3.2. File Open Dialog Box

The file open dialog box allows the user to change drives and directories and select a suitable filename of the type required from the listbox in the left middle of the dialog box. The drives and directories can be changed by double clicking on the parent directory or the drive name as seen in the respective boxes or single clicking to select and then pressing the OK button. As a result all sub directories under the double-clicked drive or directory are listed and the filenames under the sub directory with the input focus (shaded rectangle) are listed . The filename extensions are of the type shown in the type combo box on the bottom left of the dialog box. The filename is selected by either single clicking on the filename in the filename list box and then clicking the OK button or double clicking on the filename. The filename, if known can just be typed in the edit box on the top left and the OK button pressed.

The read operation starts once the filename is selected and the file successfully opened and found to be a valid .BMP file

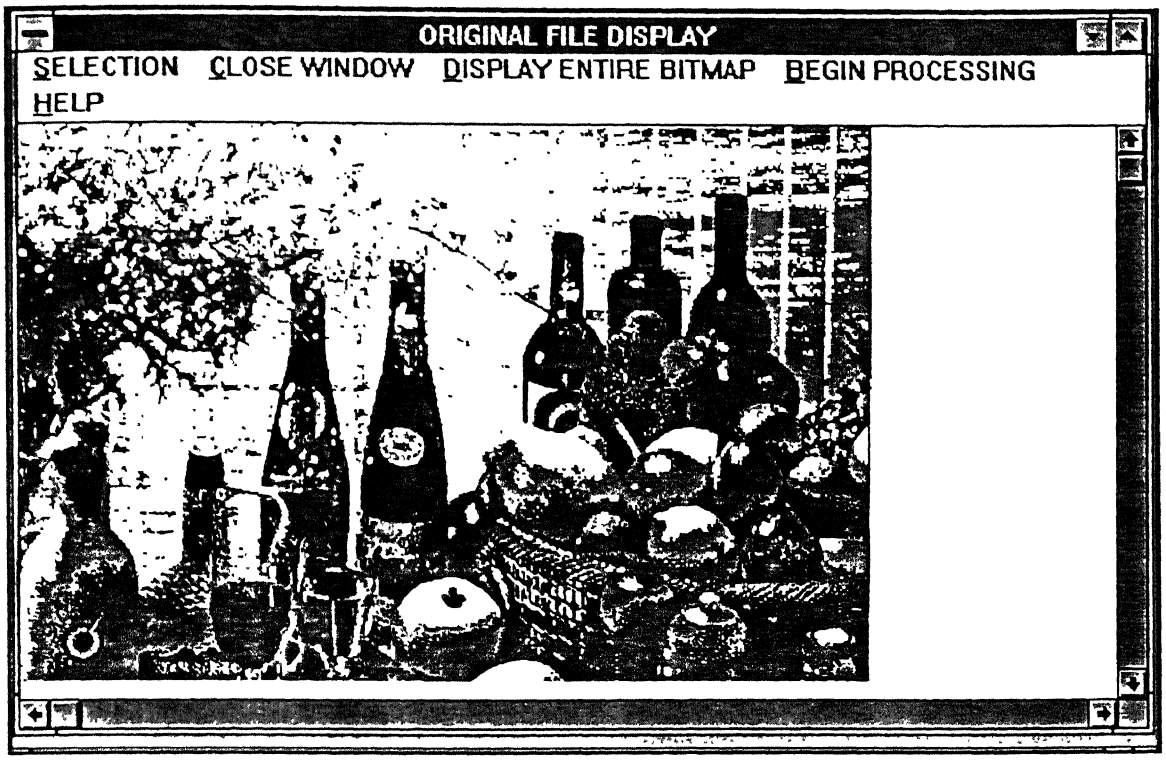


Figure 3.3. Display Window

The example.bmp file has now been opened, converted to the format given at appx 'A' and the bitmap it contains is displayed on the Display Window. The display window menu is shown along with the displayed bitmap. The SELECTION menu item allows for selection of the full bitmap or a part of it for processing. The DISPLAY ENTIRE BITMAP menu item allows the display of the full bitmap as and when desired. The BEGIN PROCESSING or the CLOSE WINDOW menu items cause the display window to minimize and control passes on to the Processing window. The image displayed in the display window can be scrolled with the help of the horizontal and vertical scroll bars on the edges of the window. This is useful for viewing a larger than screen image.

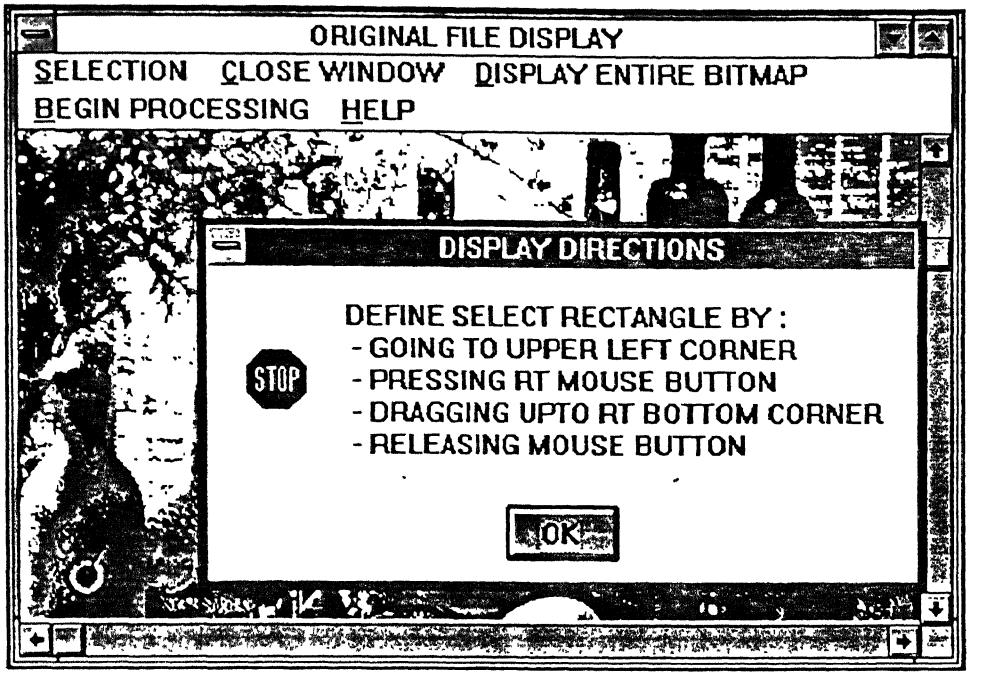


Figure 3.4. Directions For Selecting Part Of Bitmap

Fig 3.4 above shows the directions message box which comes up when the SELECTION | PART OF FILE menu item is selected. It prompts the user to earmark the area of interest with the help of the right mouse button.

Fig 3.5 ahead shows a part of the example.bmp bitmap file being selected for processing initiated with the selection of SELECT | PART OF FILE menu item . SELECT |ENTIRE FILE menu item selects the entire bitmap on display for processing. After selecting either a part or the entire bitmap, the bitmap referred to as the original bitmap is the one which comes on the screen after selection. In case of part selection a header is appended to the selected portion thus making it a separate .BMP file, ready for processing. Selecting CLOSE WINDOW or BEGIN PROCESSING menu items minimizes the display window and passes control to the processing window. The original bitmap can be viewed any time by the user by maximizing the display window or selecting the DISPLAY | SELECTED BITMAP / PORTION menu item in the processing window.

Figure 3.5. Selection Of Part Of Bitmap

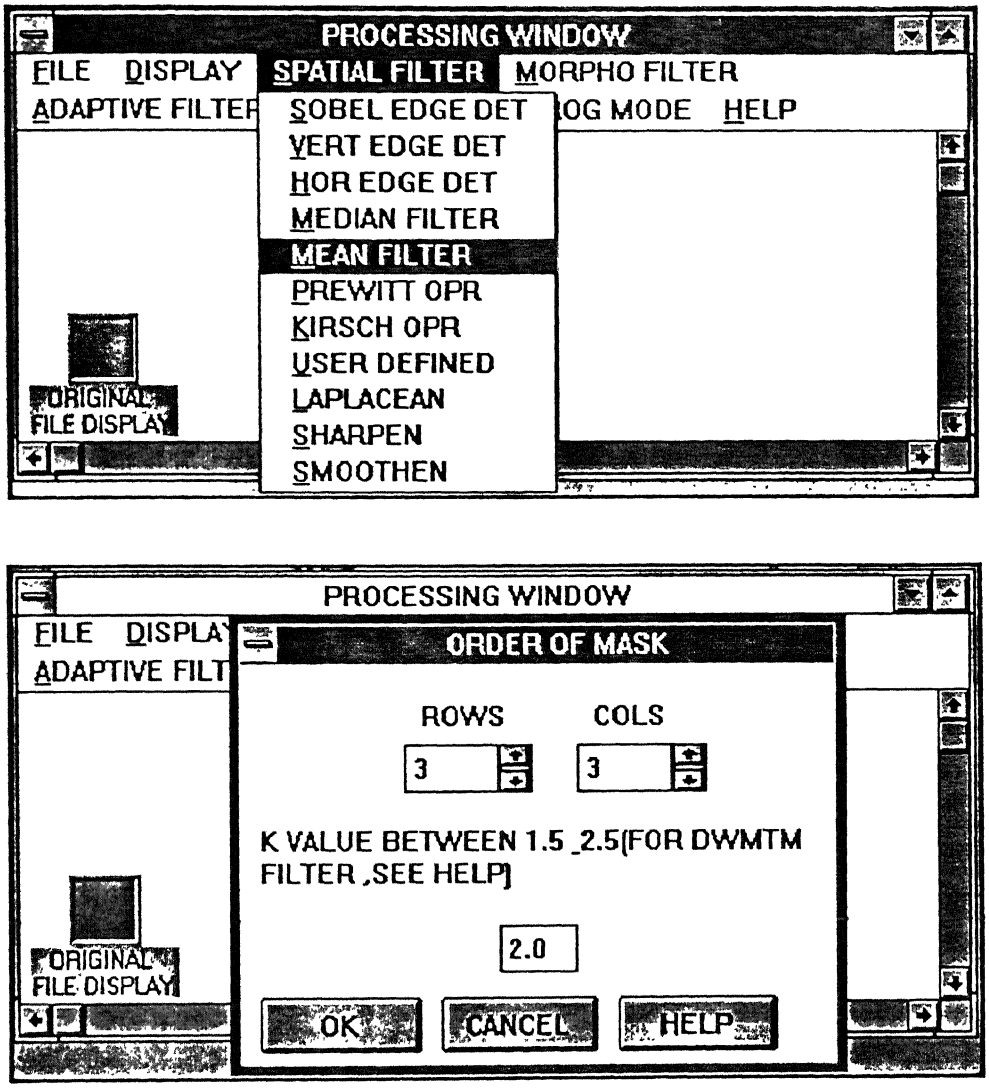


Figure 3.6. Processing The Image (Selecting The Mean Filter)

Control is now with the Processing window and the SPATIAL FILTERS | MEAN FILTER menu item has been selected for processing our bitmap. Fig 3.6 shows the mask order dialog box which comes up prompting the user to fill in the no of rows and columns required for the mean filter . This is done by scrolling the scroll bars provided with each edit box under the caption "rows" and "columns". Another edit box is provided to fill in the value of K (Ref Sec 4.3.2 DWMTM filter). This is done by replacing the already present value of K with the help of the keyboard. Once the inputs are fed in the OK

button is clicked to carry on processing. Clicking the cancel button cancels the operation and control is passed back to the processing window.

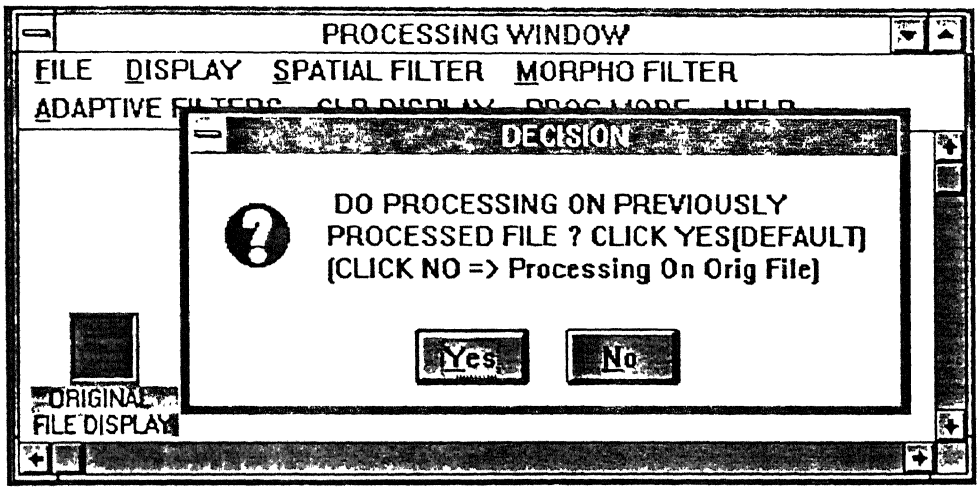


Figure 3.7. Processing Decision Message Box

A Processing decision message box prompts the user to specify which bitmap to process. It's relevance is more evident when a series of operations are being carried out . Clicking the YES button selects the previously processed bitmap for processing and NO, the original bitmap. Processing is carried out once this input is given.

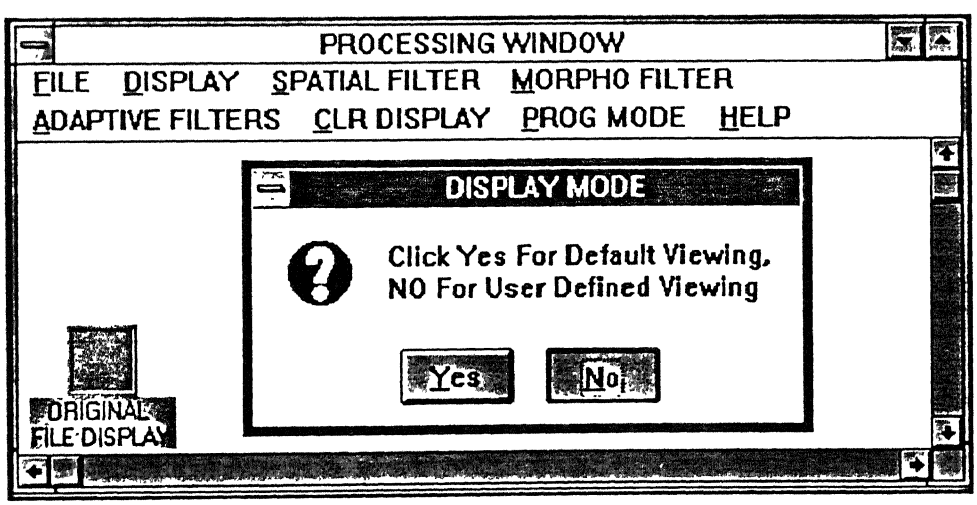


Figure 3.8. Display Decision Message Box

The above figure (fig 3.8) shows a message box asking the user to enter whether he wants to view the processed bitmap in the default mode or in the user defined mode. User defined mode is useful when it is required to retain the processed bitmap on the screen for comparison with other results. The bitmap is stretched / compressed to fit the size of the rectangle defined by the user with the help of the mouse. Scrolling facility is not available in this mode. In the default mode the bitmap is displayed in it's true size starting from pt(0,0) on the client window. Scrolling facility is available.

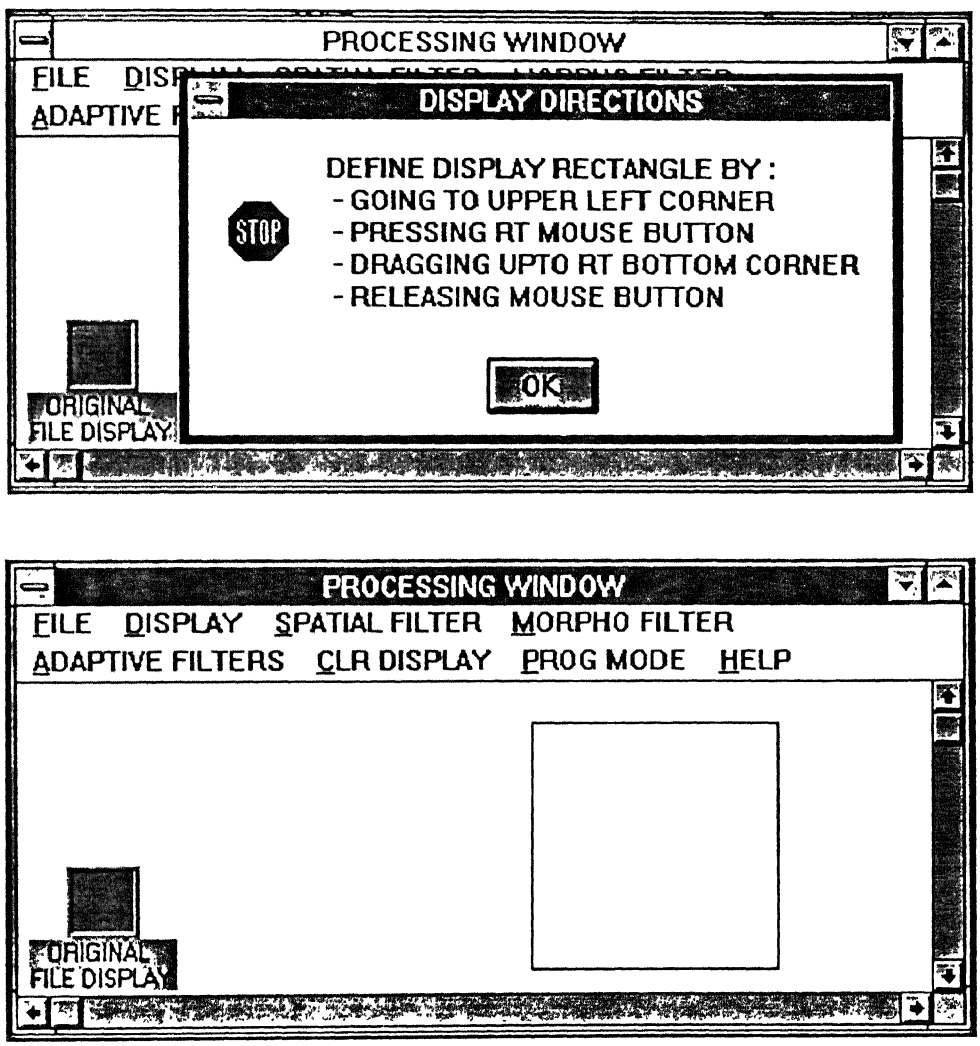


Figure 3.9. Steps To Be Taken For User Defined Display

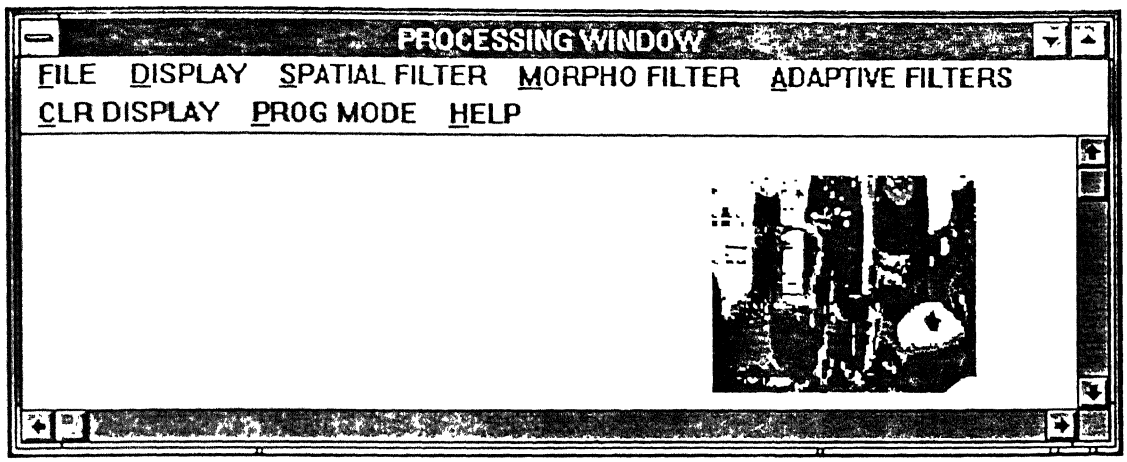


Figure 3.10. User Defined Display (Processed Bitmap)

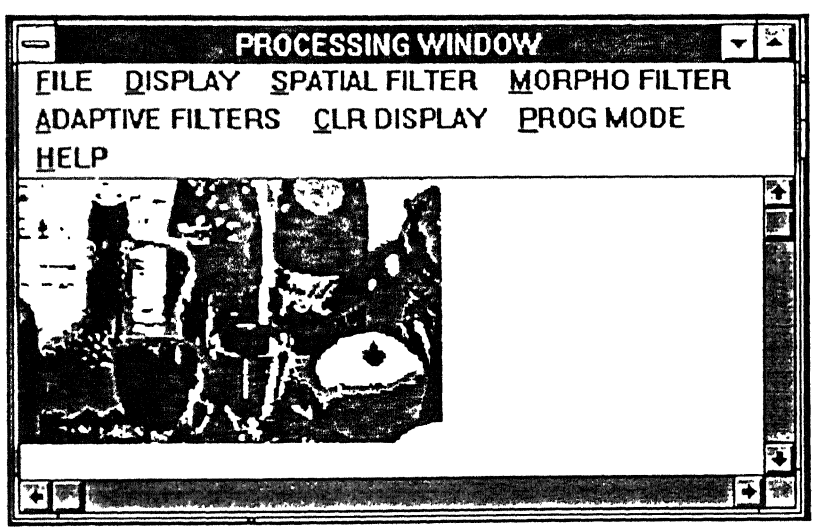


Figure 3.11. Default Display (Processed Bitmap)

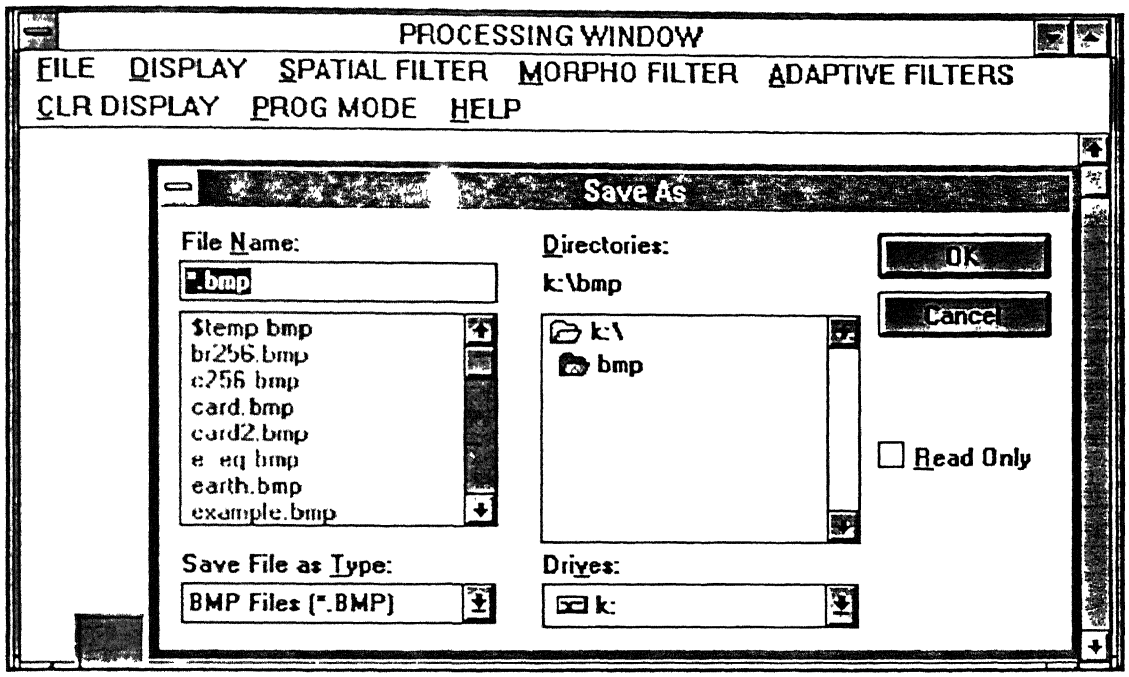


Figure 3.12. File Save Dialog Box

The processed bitmap can now, if desired, be saved by invoking the file save dialog box (fig 3.12). This is done by selecting the FILE | SAVE AS menu item. The operation of this dialog box is the same as the file open dialog box. Selecting the FILE | SAVE menu item saves the bitmap in "temp.bmp" file in the current directory. This completes one processing session. Subsequent processing is carried out in this fashion until the desired output is got.

Different types of dialog boxes are used to get the required inputs from the user. The mechanism of all these dialog boxes is more or less the same. Inputs are given in edit boxes and selections are made with the help of radio buttons. The OK, CANCEL and HELP buttons perform the same tasks in all dialog boxes. Context sensitive help can be obtained by pressing the HELP button. The inputs are canceled by the CANCEL button.

Pressing the OK button closes the dialog box and processing resumes. One of each type of these dialog boxes are shown ahead.

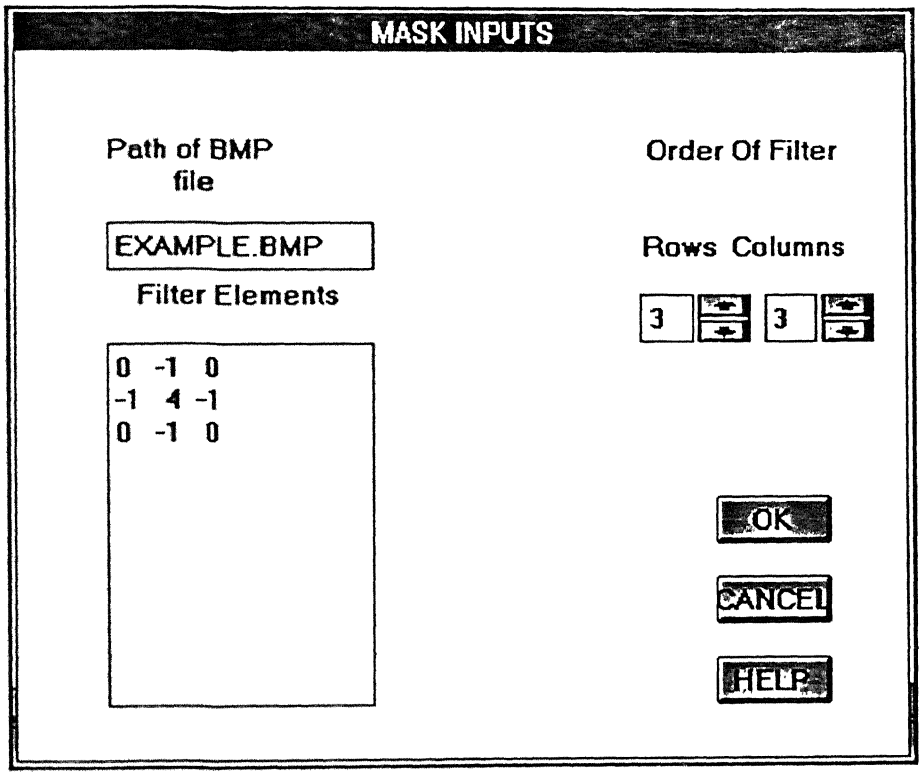


Figure 3.13. User Defined Mask Dialog Box

The user defined mask dialog box (fig 3.13) is invoked when the user is required to input the mask elements himself in addition to the order of the mask. This is done for the SPATIAL FILTER | USER DEFINED and all the MORPHOLOGICAL FILTER menu items except OUTLINING. The mask elements are entered with a space separating them. Facility exists to enter fractions also. Rest of the operations are like the mask order dialog box discussed earlier.

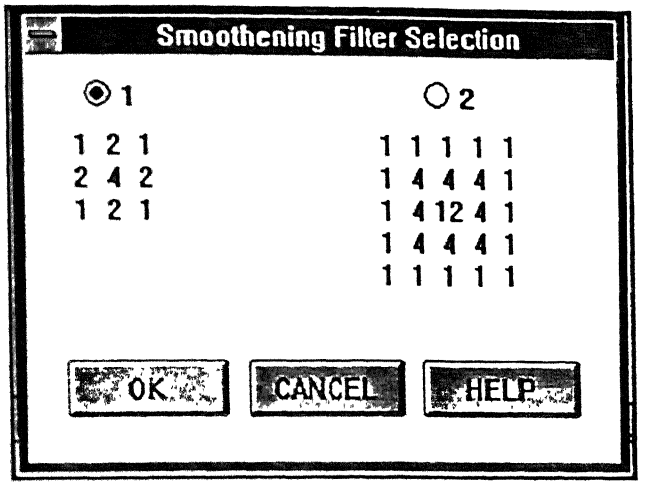


Figure 3.14. Smoothening Filter Dialog Box

Most of the SPATIAL FILTER menu items have dialog boxes allowing the user to select from a given range of masks. One such dialog box is shown in fig 3.14 for selecting a mask for smoothening operation. The selection is made by clicking on the button on top of the desired mask and the clicking the OK button.

An experiment with the batch programming mode is shown below on the example.bmp file which has already been selected earlier. The Batch mode is invoked by selecting the PROG MODE menu item from the processing window.

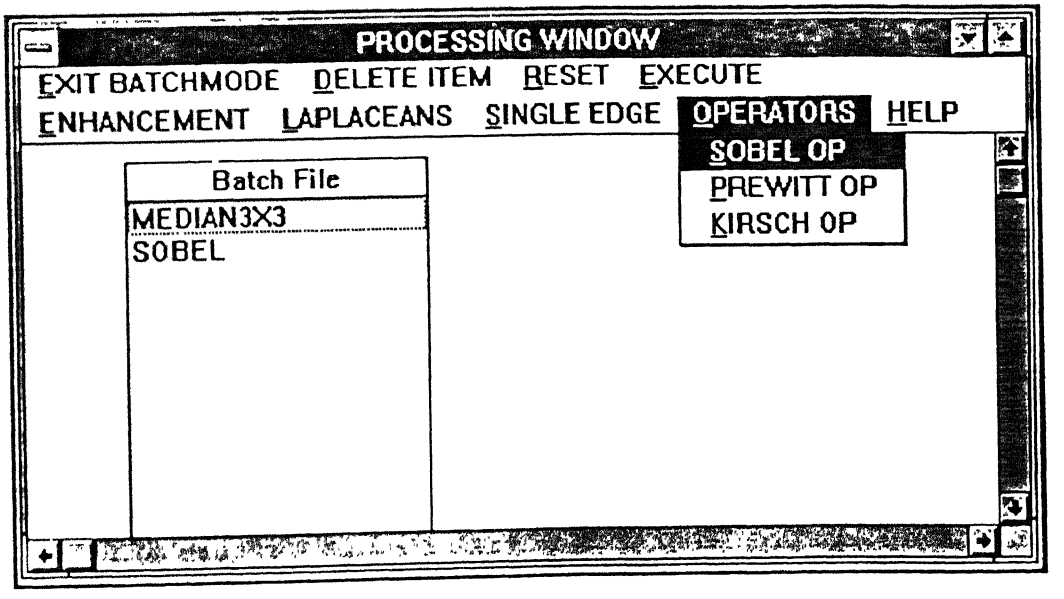


Figure 3.15. Batch Mode Window

The previous figure (fig 3.15) shows the batch mode window which has a menu bar on top allowing the user to select a series of operations to be performed on the original bitmap. Clicking the various filter and operator menu items makes them appear in the order in which they were chosen in the list box labeled "Batch File". One can delete an item from the list box by highlighting the item in the list box by a mouse click and then selecting the DELETE ITEM menu item. Insertion is also possible by highlighting the item above which another item is to be inserted and then selecting the item from the menu bar. Selecting the EXECUTE menu item applies the series of operations on the original bitmap. One can quit the Batch Mode prematurely by selecting the EXIT BATCH MODE menu item. Once the batch operations have been executed, the control automatically returns to the Processing Window and the user is prompted to make a decision about the way the processed bitmap is to be displayed (fig 3.8).

The above example was meant to highlight all the aspects of the operation of the workbench user interface as described in chapter 2.

CHAPTER 4

FILTERS : IMPLEMENTATION ASPECTS

In general a filter is a mechanism or an algorithm which modifies some aspect of an image. A wide variety of filters can be used on an image (gray level images in our context). The type of filter to be used depends upon the desired output and the characteristics of the image. The same filter when applied to two different images may produce desirable results in one and completely undesirable results in the other.

The types of filters implemented for this workbench are :

- Spatial Filters
- Morphological Filters
- Adaptive Filters

These have been dealt with separately in this chapter. All the above filters have been implemented using moving window operations. Both the image and the filter masks are considered to be 2 dimensional matrices having any arbitrary order. Odd order masks are preferred for best results as is described further. The 2 dimensional mask moves over the image pixel by pixel, starting with the first row and leftmost column, replacing the center pixel under the mask with the result of the filter computation (See fig 4.1). There is a need for defining the area of movement of the mask as is clear from the figure. Starting from the leftmost column or going upto the rightmost column would cause certain mask elements to be left out of the computation thus giving false results. An indexing scheme for accessing individual image elements falling under the mxn neighbourhood of this arbitrarily ordered mask also had to be developed to make the operations generic.

Pseudo-code for defining the centering as well as the indexing scheme is presented below . It is further elaborated in fig 4.1.

```
/*        Moving Window Operation Defined        */
for( i = no of maskrows /2 ; i < imagelength - no of maskrows /2 ; i++)
for(j = no of maskcolumns/2 ; j < imagewidth - no of maskcolumns /2 ; j++)
{
   carry out initialization;
   for ( k = 0  ; k < masklength; k++)
   for ( l = 0 ; l < maskwidth ; l++)
      /*      Indexing Scheme      */
result = Filter function output;
Center Pixel Value = result;
/* Where Centre pixel is defined as :
*( 0th Image element +  (i - no of maskrows/2 + k) x image width + j - no of
maskcolumns / 2 + l)                                                    */
}
```

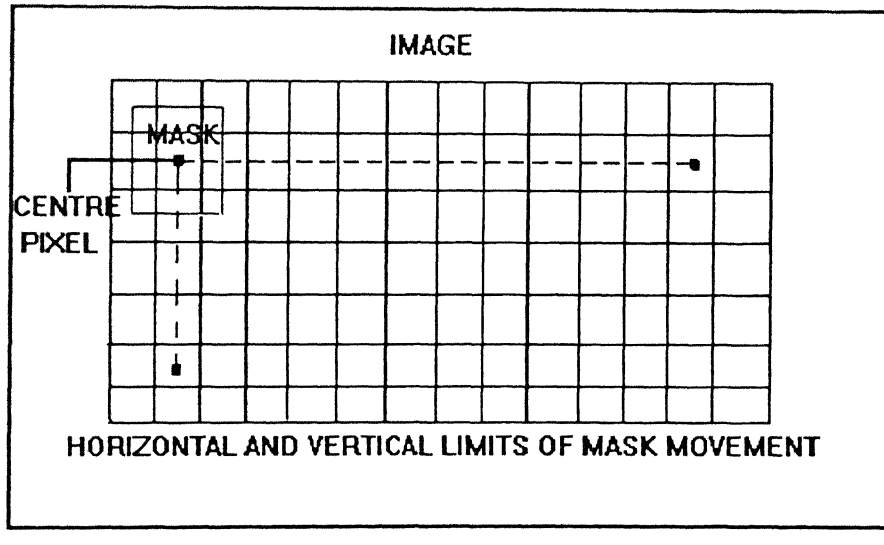


Figure 4.1. Moving Window Operation

From fig 4.1 it is evident that the mask movement is limited by the no of rows and columns it contains. As a result a border of (no of maskrows/2 x no of maskcolumns/2 pixels) is left unprocessed. In order to define centering more accurately it is advisable to use odd ordered masks. The division here is integer division.

4.1 Spatial Filters

Spatial filters are basically discrete convolution filters convolving the image with the filter mask Convolution is a mapping from an image to another image whose pixel(i,j) in the range image is some function (usually a weighted sum) of pixel(i,j) in the domain image along with neighbouring pixels decided by the convolution mask. Based on the size and weights in the mask the resulting image has certain features. For example high/low pass filtering, vertical or horizontal edge detection, detection of edges in any orientation etc.

Spatial filters incorporate local, or neighbourhood operations on pixels. In other words the computations affecting a particular pixel in an image involve the value of that pixel and it's neighbouring pixels and not pixels elsewhere in the image. Computations for a wide variety of interesting filter operations have been accomplished using nothing more than the convolution operation described above. Depending on the operation the user has the facility to select from a variety of standard masks available. A facility for the user to use his own mask is also provided. The user can enter this mask in the form of integers or fractions in the user defined mask dialog box (fig 3.13). This dialog box processing function reads in the input as a string and then parses the input as integers in case there are no '/' characters between two subsequent numbers. Even if one '/' character is encountered the LCM of all denominators of elements (now considered as fractions) is calculated and stored. The numerators are multiplied by the quotient of the LCM and the

respective denominator and entered as mask elements. The convolution results for every pixel are later divided by the denominator LCM while scaling the result to a range of 0 to 255.

The filter output is the result of the convolution of the selected mask with the image. The output is available in the form of a huge pointer to BYTE. It contains all the header information required and can straight-away be saved as a .BMP file or used for further processing. The following spatial filters have been implemented based on the technique discussed above :

- Laplacian filter
- Sharpening filter
- Smoothening filter
- Mean filter
- Median filter
- Sobel edge detector
- Prewitt edge detector
- Kirsch operator

See appx 'B' for the masks used in the above.

4.2 Morphological Filters

Morphological filters operate on the objects within an image by manipulating an object's geometrical shape. Usually the final goal of image processing is to segment the objects within an image for object recognition and identification. The image processing steps required prior to pattern recognition and identification usually include spatial filtering of the image to remove noise, then thresholding of the graylevel image to produce a binary image and finally morphological filtering for geometrical shape decomposition.

Morphological filters have been implemented in this workbench for binary images. All of them use the moving window operation described earlier in this chapter. The goal

of these filters is to smooth the contours of the objects and to decompose an image into it's fundamental geometrical shapes. A brief description of the implementation of the morphological filters is given below. The algorithms are based on the concepts of set theory and Minkowski algebra. For more details refer to [CImRC].

4.2.1 Morphological Dilation And Erosion Filters

Morphological operations implemented here are functions of two set operations, Minkowski set addition and substraction. These are defined below with the help of fig 4.2 and fig 4.3. Fig 4.2 shows two objects A and B located in a two dimensional space R^2 . and B both are rectangular objects. B is symmetric about the origin.

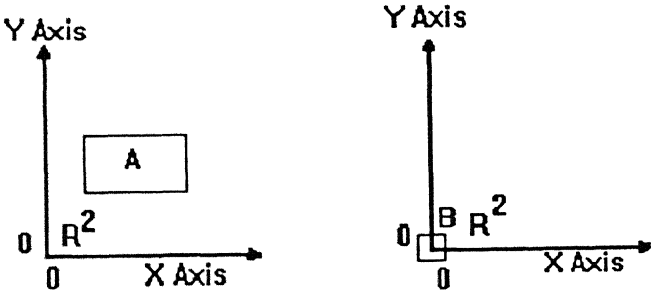


Figure 4.2. Two Objects A and B Shown In Two Dimensional Space R^2

The Minkowski addition of the two sets A and B given by the symbol $\oplus$ is

$$A \oplus B = \{ t \in R^2 : t = a + b , a \in A , b \in B \} \quad (1)$$

Minkowski substraction of the above sets A and B given by the symbol Θ is

$$A \Theta B = (A^c \oplus B)^c \quad (2)$$

These are shown diagramatically in fig 4.3 below.

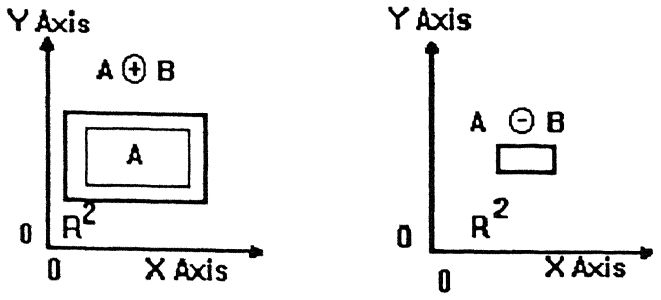


Figure 4.3 Minkowski Addition and Subtraction

Morphological dilation is nothing but the Minkowski addition of the image with a structuring set symmetric about it's origin.

$$\text{Result} = \text{Image} \oplus \text{Dilation mask.}$$

The dilation operation uses a maximum filter. The image pixels below the mask region with mask values equal to one are used to compute a maximum value. Since the images in question are binary images, the possible outputs of the dilation filter are 0 or 255. The pixel value below the center mask element is replaced by this maximum value. This process is repeated for all the pixels within the image falling under the center element of the mask . The mask in this case is specified by the user in the "user defined mask dialog box " (See fig 3.13) along with the number of rows and columns.

Erosion can be described in terms of Minkowski algebra as the Minkowski subtraction of the Erosion mask (structuring set symmetric about the origin) from the Image. It can be described in terms of Minkowski addition as both are duals.

$$\text{Result} = (\text{Image}^c \oplus \text{Erosion Mask})^c$$

Fig 4.3 serves as an example for dilation and erosion as well because set B is symmetric about the origin.

The erosion filter is implemented in the same way as above. The only difference is that a minimum filter is used to compute the minimum value from the pixels under the mask with mask elements equal to 1. This value replaces the value of the pixel under the center mask element.

4.2.2 Morphological Opening And Closing

Morphological opening and closing operation have been implemented using the erosion and dilation filters described above. These are used to geometrically filter the contours of objects. Small narrow openings and small outward bumps can be reduced by an opening operation. The closing operation on an object will fill small inward bumps and small holes in an object's contour. Thus opening and closing operations result in new geometrical objects in which contours have been filtered.

Morphological opening is erosion followed by dilation and is given by :

Open(Image , Mask) = Dilation (Erosion (Image , Mask) , Mask)

Morphological closing is dilation followed by erosion and is given by :

Close (Image , Mask) = Erosion(Dilation (Image , Mask) , Mask)

Mask elements have to be specified by the user as in 4.2.1 above after which the erosion and dilation routines are called as required to implement the above operations. The output is available as in the case of spatial filters as a huge pointer to BYTE.

4.2.3 Morphological Outlining

Morphological outlining or contour detection is the process of removing all of an object's pixel information except that present on the object's contour. The advantage of using a morphological contour detector vis-a-vis a spatial edge detector is that it is more immune to noise.

A one pixel wide contour detector has been implemented for this workbench . The image is first eroded with a standard mask (See appx ' B') and then the resultant image is subtracted from the original image to give the desired output.

4.3 Adaptive Filters

The filters discussed in the previous sections operate either locally or globally on an image with their filtering characteristics remaining constant throughout. The disadvantage of this type of filtering is that some important details may be removed in the process as

the local characteristics of an image change from one part to another. Thus there is a requirement of filters which can change their filtering characteristics depending on the image content within a local window. For example, if the noise within an image has a uniform distribution, it is best filtered using a mean filter, but at the expense of losing some fine spatial details and blurred edges. These are preserved if a median filter is used, but it is best for removal of impulsive-type noise. An ideal filter to use on an image would thus be such that if the image contents within a local window contained edges, a median filter would be used and if the image contents within the local window contained a uniform background, a mean filter would be used. Thus the final goal of an adaptive filter is to change it's characteristics to best preserve the image contents and at the same time reduce the noise within the image.

A brief description of the implementation of two adaptive filters implemented for this workbench follows. Both these filters require noise variance as an input. Noise variance is nothing but the variance of an area of the noise corrupted image which contains no edges but only the background information or noise . This area has to be specified by the user with the help of a mouse defined rubber banded rectangle in the same way as the area of the image required for processing was specified (see fig 3.9).

4.3.1 MMSE Filter

The minimum mean-square error filter makes use of the knowledge of the local variance to determine if a mean filter is to be applied to the local region . The comparison of the noise variance (σ_n)and the local variance (σ_l) is done to make the filter decision which can be summarized best as :

$$\text{Result} = \left(1 - \frac{\sigma_n^2}{\sigma_l^2}\right) \times \text{Image} + \frac{\sigma_n^2}{\sigma_l^2} \times (\text{out put from the local mean filter}) \qquad (3)$$

In the background region, the only variations in the pixel values are due to the noise alone. Computing the local variance in this region will yield a value that is approximately equal to the noise variance. In this case $\sigma_n^2/\sigma_l^2 \approx 1$. The filtered image output thus reduces to the output of the mean filter. The order of the mean filter is given by the user in the mask order dialog box (See fig 3.6). As the local region moves into an area within the image which contains an edge, the local variance will become much larger than the noise variance due to the edge. Hence $\sigma_l^2 >> \sigma_n^2$ and therefore the output of the MMSE filter from equation 3 reduces to the original unfiltered image being used as the input . Hence the MMSE filter will be able to filter the noise without blurring any edges. However this filter fails in case of impulse-type noise like salt and pepper noise. This is because the mean filter does not do any filtering in case local variance is much greater than the noise variance. The presence of a noise outlier within a local region will make the local variance much greater even if no edge is present. To take care of this one makes use of the DWMTM filter described in the next subsection.

4.3.2 DWMTM Filter

The Double Window Modified Trimmed Mean filter overcomes the difficulty of using the MMSE filter in the presence of impulsive noise by using the median estimator to estimate the local mean. Then a new local mean is calculated using pixels within only a small graylevel range about the median. This effectively eliminates the noise outliers in the calculation of the local mean and variance.

Initially, an mxn order mask is assumed given the order in the mask order dialog box (See fig 3.6) . The median of an mxn neighbourhood about each pixel in the image is calculated. The median value from this calculation is used now to calculate a new local mean, this time in a bigger neighbourhood (m+2) x (n+2) . But in this computation only pixels within a graylevel range of

Median[mxn] -c to Median[mxn] + c

are taken into consideration. The value of c is given by

$$c = K \times \sigma_n$$

Typical values of K range from 1.5 to 2.5 . This is assumed to be 2.0 by default unless changed by the user in the mask order dialog box (See fig 3.6) . Thus the output of the filter is nothing but the local mean calculated by the above criterion .

CHAPTER 5

SUMMARY AND CONCLUSIONS

The objective of designing an image processing workbench for filter functions has been carried out keeping in mind , the scope of the thesis . Processing results on various images by the filters implemented in this workbench are available at appx 'C'.

5.1 Summary Of Work Done

The image display has been made device independent by using the inbuilt capability of the Windows API functions to map the bitmap colors to the nearest available color in the system palette. The palette can be varied statistically depending on the capability of the device and whether it supports palette operations, thus making the display as faithful as possible. The workbench can handle large images whose size is limited to 1/5 of the free disk space or 1/5 of 16MB whichever is less. This is so because more than one buffer is required to store the results of intermediate processing.

The user interface has been designed to make the user interaction with the workbench as natural as possible . Most of the inputs can be given by the user with the help of the mouse. Flexibility has been ensured by giving the user, control over the package in the form of the various user tools provided. Amongst the tools provided for client area manipulation and image handling, the batch programming tool is the most useful. This results in a lot of time saving as no inputs are required in between the processing sequence.

Most of the image processing, file handling and bitmap handling functions used in the workbench are available for further use and modification in the form of a DLL called "imajdll.lib" . The workbench program has been compiled and linked using the Borland C++ version 4.0 compiler and linker, in the large memory model .

5.2 Recommendations For Further Work

The workbench is a first step in the development of a full fledged image processing system which integrates acquisition and display, processing and recognition or understanding. The message driven architecture of windows is ideally suited for this kind of a job. In order to expand the scope of this package to an image processing system, the "image acquisition" functionality needs to be backward integrated into this package. Compatible drivers for CCD cameras and scanner operations need to be incorporated into the system. Secondly, forward integration with "image understanding and interpretation" software needs to be carried out. The processed images from this workbench can serve as inputs to the above .

The batch programming facility could be expanded into an interpreter which could include control and decision loops. The programs written for such an interpreter from within the system could be stored under a name to make them easier to be recalled for later use. Thus remembering an often used sequence of operations to be performed on an image everytime can be done away with.

Last but not the least, routines for file format conversions are required to be built into the package, thus enhancing it's capability to accept inputs in any format. Similarly saving could also be implemented in the desired format. This would save a lot of disk space by storing and accepting images in compressed formats.

BIBLIOGRAPHY

[EVGA] Bradley Dick Kleiwer , *EGA / VGA A Programmer's ReferenceGuide,* Intertext Pub (2nd Ed), 1990.

[CharP] Charles Petzold, *Programming Windows 3.1,* Microsoft Press, 1992.

[GrpFF] David C Kay, John R Lewine , *Graphics File Formats* , Windcrest / Mc Graw Hill , 1993.

[MatIm] Edward R Doughetry, Charles Giardina , *Matrix Structured Image Processing*, Prentice Hall Inc, 1987.

[CImRC] Harley R Myler, Arthur R Weeks, Computer Imaging Recipes in C , PTR Prentice-Hall Inc, 1993.

[WAPI] James L Conger, *Windows API Bible,* Waite Group Inc, 1992.

[ITDIP] Niblack W, *Introduction To Digital Image Processing* ,Prentice Hall (UK) Ltd , 1986.

GLOSSARY

Child Window Controls : Child window controls are predefined / user defined controls which act as input devices for their parent windows . These controls take the form of buttons , check boxes , edit boxes , list boxes , combo boxes scroll bars etc. . These are most commonly used in dialog boxes . They can also be used independently . Although one can create one's own child window controls , one can also take advantage of certain predefined window classes along with their window processing functions .

Client Area : Client area is the part of the window on which a program is free to draw . The client area occupies all the space of the window that is not taken up by the caption bar , border , menu bar and scroll bar (if any) .The size of the client area varies with the window size .

Device Context : A device context refers to a physical output device (such as a video display) and it's device driver . The device context contains the various attributes that determine how the GDI functions work on the device . These attributes include the color of text , background color , font , inter character spacing The handle to the device context is obtained and passed as a parameter to the GDI functions .

Dialog Boxes : A dialog box is most often used for obtaining additional input from the user beyond what can be easily managed through a menu . It generally takes the form of a popup window containing various child window controls . When the dialog box has the input focus , all messages are sent to the window processing function of the dialog box , rather than that of the parent window .

GDI : The Graphics Device Interface is a module (GDI.EXE) which contains functions handling all graphics related tasks in Windows . The functions in this module also call routines in the driver files (.DRV) which in turn access the hardware of the respective devices . One of the primary aims of GDI is to support device independent graphics on output devices such as video displays , printers and plotters .

GUI : Windows is a Graphical User Interface or a visual interface or a graphical windowing environment . All GUI's make use of graphics on a bitmapped video display . Graphics provides better utilization of screen real estate , a visually rich interface for conveying information , and the possibility of a WYSIWYG (What you see is what you get) video display of graphics and formatted text prepared for a printed document .

Input Focus : A window with the input focus is either an active window or a child window of the active window . It is the window processing function of the active window which receives all the messages till the time it , or one of it's child windows retain the input focus .

Maximize : To maximize a window is to make it occupy the entire screen . This is done by clicking on the maximize button on the right of the caption bar or through the control menu .

Memory Device Context : A memory device context is a block of memory which is created over the display device context for the purpose of drawing on it . It has a display surface that exists only in memory . Once a bitmap is selected into the

memory device context , it's display surface size becomes equal to the bitmap . The memory device context has it's attributes identical to the display device context .

Message : Windows communicates with the programs running under it with the help of messages . When Windows sends a message to a program , it calls the Window processing function of the active window in that program . The parameters of that function describe the message . The window processing function does some processing based on the message and returns control to Windows .

Message Box : The message box is a special kind of a dialog box which is used as it's alternative when the response expected from the user is a simple one ie. pressing of a button out of two or three standard buttons available in the message box . The message box appears in response to a standard API function and no separate Window processing function is required to be written for it .

Message queue : Windows maintains a message queue for each Windows application currently running under it . When an input event occurs , Windows translates it into a message and places it in the message queue . A program retrieves messages from the message queue by executing a block of code known as a message loop .

Minimize : To minimize a window is to convert it into an icon by either clicking on the minimize button situated on the right of the caption bar and left of the maximize / restore button or through the control menu .

Restore : To restore a window is to bring it back to it's original size when it was created . This is done by clicking on the restore button on the right of the minimize button . This button is also the maximize button when the window is in it's original size . It

becomes the restore button when the window is maximized . The same can be accomplished through the control menu

Window Class : A window is always created based on a window class . The window class identifies the window processing function that processes the messages to that window and some other characteristics of the window that are created based on the class . More than one window can be created based on a single class .

Window Processing Function : The window processing function or the window procedure is a function which processes the messages sent by Windows . Each window is associated with only one window procedure at a particular time .

APPENDIX A
FILE FORMATS AND DATA STRUCTURES USED

BITMAPFILEHEADER DATA STRUCTURE
BITMAPINFOHEADER DATA STRUCTURE
COLOR TABLE CONSISTING OF RGBQUAD DATA STRUCTURES
PIXEL DATA

BITMAPFILEHEADER Data Structure

```
typedef struct tagBITMAPFILEHEADER {
    WORD    bfType;
    DWORD   bfSize;
    WORD    bfReserved1;
    WORD    bfReserved2;
    DWORD   bfOffBits;
} BITMAPFILEHEADER;
```

The BITMAPFILEHEADER data structure contains information about the type, size, and layout of a file that contains a device-independent bitmap (DIB).

Member Description

- bfType Specifies the file type. It must be BM.
- bfSize Specifies the size of the file, in a double word value. This value can be obtained by dividing the length of the file, in bytes, by 4.
- bfReserved1 Reserved; must be zero.
- bfReserved2 Reserved; must be zero.
- bfOffBits Specifies the offset, in bytes, from theBITMAPFILEHEADER to the bitmap bits.

BITMAPINFO Data Structure

```
typedef struct tagBITMAPINFO {
  BITMAPINFOHEADER   bmiHeader;
  RGBQUAD            bmiColors[1];
} BITMAPINFO;
```

The BITMAPINFO structure fully defines the dimensions and color information for a Windows 3.0 or later device-independent bitmap (DIB).

Member Description

- bmiHeader Specifies a BITMAPINFOHEADER structure that contains information about the dimensions and color format of a DIB.
- bmiColors Specifies an array of RGBQUAD structures that define the colors in the bitmap.

A Windows 3.0 or later DIB consists of two distinct parts: a BITMAPINFOstructure, which describes the dimensions and colors of the bitmap, and an array of bytes defining the pixels of the bitmap. The bits in the array are packed together, but each scan line must be zero-padded to end on a LONG boundary.. The origin of the bitmap is the lower-left corner.

BITMAPINFOHEADER Data Structure

```
typedef struct tagBITMAPINFOHEADER {
  DWORD  biSize;
  LONG   biWidth;
  LONG   biHeight;
  WORD   biPlanes;
  WORD   biBitCount;
  DWORD  biCompression;
  DWORD  biSizeImage;
  LONG   biXPelsPerMeter;
  LONG   biYPelsPerMeter;
  DWORD  biClrUsed;
  DWORD  biClrImportant;
} BITMAPINFOHEADER;
```

The BITMAPINFOHEADER structure contains information about the dimensions and color format of a Windows 3.0 or later device-independent bitmap (DIB).

Member Description

- biSize: Specifies the number of bytes required by theBITMAPINFOHEADER structure.
- biWidth : Specifies the width of the bitmap, in pixels.
- biHeight : Specifies the height of the bitmap, in pixels.
- biPlanes : Specifies the number of planes for the target device.
- biBitCount : Specifies the number of bits per pixel..

  Value Meaning

  1 The bitmap is monochrome, and the bmciColors member must contain two entries. Each bit in the bitmap array represents a pixel.If the bit is clear, the pixel is displayed with the color of the first entry in the bmciColors table. If the bit is set, the pixel has the color of the second entry in the table.

  4 The bitmap has a maximum of 16 colors, and the bmciColors member contains 16 entries. Each pixel in the bitmap is represented by a four-bit index into the color table.

  8 The bitmap has a maximum of 256 colors, and the bmciColors member contains 256 entries. In this case, each byte in the array represents a single pixel.

  24 The bitmap has a maximum of 2^24 colors. The bmiColors member is NULL, and each 3-byte sequence in the bitmap array represents the relative intensities of red, green, and blue, respectively, of a pixel.

- biCompression : Specifies the type of compression for a compressed bitmap. It can be one of the following values:

Value Meaning

- BI_RGB : Specifies that the bitmap is not compressed.
- BI_RLE8 : Specifies a run-length encoded format for bitmaps with 8 bits per pixel.
- BI_RLE4 : Specifies a run-length encoded format for bitmaps with 4 bits per pixel.

- biSizeImage : Specifies the size, in bytes, of the image. It is valid to set this member to zero if the bitmap is in the BI_RGB format.
- biXPelsPerMeter Specifies the horizontal resolution, in pixels per meter, of the target device for the bitmap.
- biYPelsPerMeter Specifies the vertical resolution, in pixels per meter, of the target device for the bitmap.
- biClrUsed Specifies the number of color indexes in the color table actually used by the bitmap. If this value is zero, the bitmap uses the maximum number of colors corresponding to the value of the biBitCount member . If the biClrUsed member is nonzero, it specifies the actual number of colors that the graphics engine or device driver will access if the biBitCount member is less than 24. If biBitCount is set to 24, biClrUsed specifies the size of the reference color table used to optimize performance of Windows color palettes.
- biClrImportant Specifies the number of color indexes that are considered important for displaying the bitmap. If this value is zero, all colors are important.

RGBQUAD Data Structure

```
typedef struct tagRGBQUAD {
  BYTE   rgbBlue;
  BYTE   rgbGreen;
  BYTE   rgbRed;
  BYTE   rgbReserved;
} RGBQUAD;
```

The RGBQUAD structure describes a color consisting of relative intensitiesof red, green, and blue.

For the purpose of this thesis the standard format used for bitmap storage and processing has the following values fixed

- The biBitCount member of the BITMAPINFOHEADER is 8
- The biClrUsed member value of the BITMAPINFOHEADER is 0 .
- The biClrImportant member value of the BITMAPINFOHEADER is0.
- The biCompression member value of the BITMAPINFOHEADER is BI_RGB implying an uncompressed bitmap .
- The color table values range from 0,0,0 to 255,255,255 for Red ,Green and Blue members in the RGBQUAD data structure . The color table has 256 members .

The rest of the member values are set according to the bitmap characteristics .

The OPENFILENAME Data Structure

```
typedef struct tagOFN {
        DWORD       lStructSize;
        HWND        hwndOwner;
        HINSTANCE hInstance;
        LPCTSTR     lpstrFilter;
        LPTSTR      lpstrCustomFilter;
        DWORD       nMaxCustFilter;
        DWORD       nFilterIndex;
        LPTSTR      lpstrFile;
        DWORD       nMaxFile;
        LPTSTR      lpstrFileTitle;
        DWORD       nMaxFileTitle;
        LPCTSTR      lpstrInitialDir;
        LPCTSTR     lpstrTitle;
        DWORD       Flags;
        WORD        nFileOffset;
        WORD        nFileExtension;
        LPCTSTR     lpstrDefExt;
        DWORD       lCustData;
        LPOFNHOOKPROC lpfnHook;
        LPCTSTR     lpTemplateName;
        } OPENFILENAME;
```

The OPENFILENAME structure contains information the operating system uses to initialize the system-defined Open or Save As dialog box. After the user closes the dialog box, the system returns information about the user's selection in this structure.

Member Description

- lStructSize : Specifies the length, in bytes, of the structure.
- hwndOwner : Identifies the window that owns the dialog box. hInstance Identifies a data block that contains a dialog box template specified by the lpstrTemplateName member.
- lpstrFilter : Points to a buffer containing pairs of null-terminated filter strings. The first string in each pair describes a filter (for example, "Text Files"),the second specifies the filter pattern (for example, "*.TXT").
- lpstrCustomFilter : Points to a buffer containing a pair of user-defined filter strings. The first string describes the filter, and the second specifies the filter pattern
- nMaxCustFilter : Specifies the size, in characters, of the buffer identified by the lpstrCustomFilter member.
- nFilterIndex : Specifies an index into the buffer pointed to by the lpstrFilter member.
- lpstrFile : Points to a buffer that contains a filename used to initialize the File Name edit control.
- nMaxFile : Specifies the size, in characters, of the buffer pointed to by the lpstrFile member.
- lpstrFileTitle : Points to a buffer that receives the title of the selected file
- nMaxFileTitle : Specifies the maximum length of the string that can be copied into the lpstrFileTitle buffer.
- lpstrInitialDir : Points to a string that specifies the initial file directory. If this member is NULL, the system uses the current directory as the initial directory.

- lpstrTitle : Points to a string to be placed in the title bar of the dialog box.
- Flags : Specifies the dialog box creation flags.
- nFileOffset : Specifies a zero based offset from the beginning of the path to the filename in the string to which the lpstrFile member points .
- nFileExtension : Specifies a zero-based offset from the beginning of the path to the filename extension in the string pointed to by the lpstrFile member.
- lpstrDefExt : Points to a buffer that contains the default extension .
- lCustData : Specifies application-defined data that the operating system passes to the hook function identified by the lpfnHook member.
- lpfnHook : Points to a hook function that processes messages intended for the dialog box.
- lpstrTemplateName : Points to a null-terminated string that names the dialog box template resource to be substituted for the standard dialog box template. Default NULL .

APPENDIX B

STANDARD MASKS USED

Sobel Operator

mask(x) mask(y)

```
-1  0  1    -1 -2 -1
-2  0  2     0  0  0
-1  0  1     1  2  1
```

Prewitt Operator

mask(x) mask(y)

```
-1  0  1    -1 -1 -1
-1  0  1     0  0  0
-1  0  1     1  1  1
```

Horizontal Edge Detector

```
-2  -2  -2
 0   0   0
 2   2   2
```

Vertical Edge Detector

```
-2  0  2
-2  0  2
-2  0  2
```

Laplacian Masks

LAP3

```
 0  -1  0
-1   4 -1
 0  -1  0
```

LAP5

```
-1  -1  -1  -1  -1
-1  -1  -1  -1  -1
-1  -1  24  -1  -1
-1  -1  -1  -1  -1
-1  -1  -1  -1  -1
```

LAP9

```
-1  -1  -1  -1  -1  -1  -1  -1  -1
-1  -1  -1  -1  -1  -1  -1  -1  -1
-1  -1  -1  -1  -1  -1  -1  -1  -1
-1  -1  -1   8   8   8  -1  -1  -1
-1  -1  -1   8   8   8  -1  -1  -1
-1  -1  -1   8   8   8  -1  -1  -1
-1  -1  -1  -1  -1  -1  -1  -1  -1
-1  -1  -1  -1  -1  -1  -1  -1  -1
-1  -1  -1  -1  -1  -1  -1  -1  -1
```

Sharpening Filters

SHARP3

```
-1  -1  -1
-1   9  -1
-1  -1  -1
```

SHARP5

```
 0  -1   1  -1   0
-1   2  -4   2  -1
-1  -4  13  -4  -1
-1   2  -4   2  -1
 0  -1   1  -1   0
```

Smoothening Filters

SMOOTH3

```
1  2  1
2  4  2
1  2  1
```

SMOOTH5

```
1  1  1  1  1
1  4  4  4  1
1  4 12  4  1
1  4  4  4  1
1  1  1  1  1
```

Mask Used for Outlining Operation

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

APPENDIX C
RESULTS

This appendix gives the results of processing done with the filters implemented in this workbench .

Figure C.1. Original Image (Girl's Face)

Figure C.2 Sobel Operator Applied To Girl's Face Image

Figure C.3 Prewitt Operator Applied To Girl's Face Image

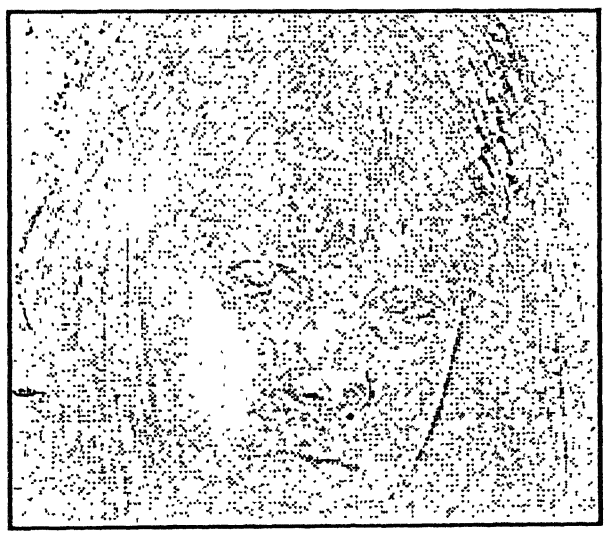


Figure C.4 3x3 Laplacian Mask Applied To Girl's Face Image

Figure C.5 3x3 Sharpen Operator Applied To Girl's Face Image

Figure C.6 Original Image (Wine)

Figure C.7 3x3 Median Filter On Wine Image

Figure C.8 3x3 Mean Filter On Wine Image

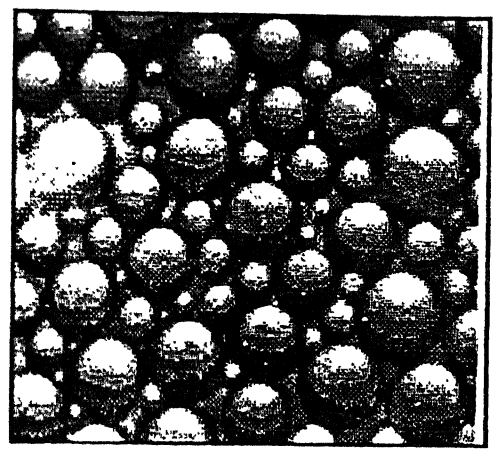


Figure C.9 Original Image (Histogram Equalized Image Of A Metal Surface)

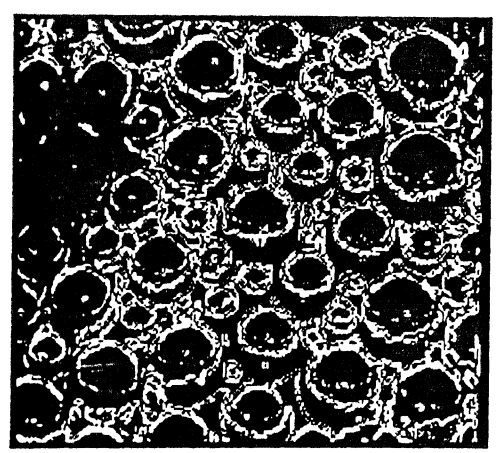


Figure C.10 Edge Detected Metal Surface Image

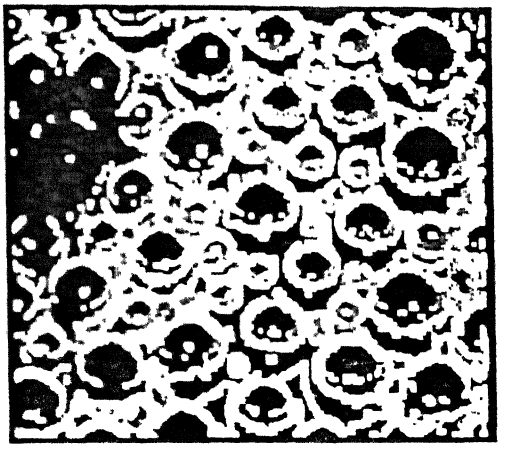


Figure C.11 Dilation Of Image In Fig C.10

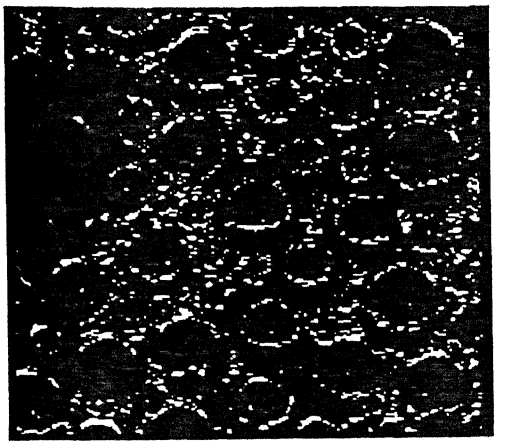


Figure C.12 Erosion Of Image In Fig C.10

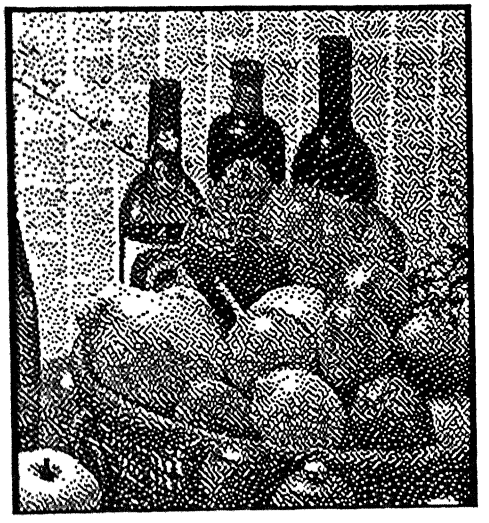


Figure C.13 Original Image (Dithered Bi-Level Wine Image)

Figure C.14 MMSE Filter On Image In Image Fig C.13

Figure C.15 DWMTM Filter On In Fig C.13